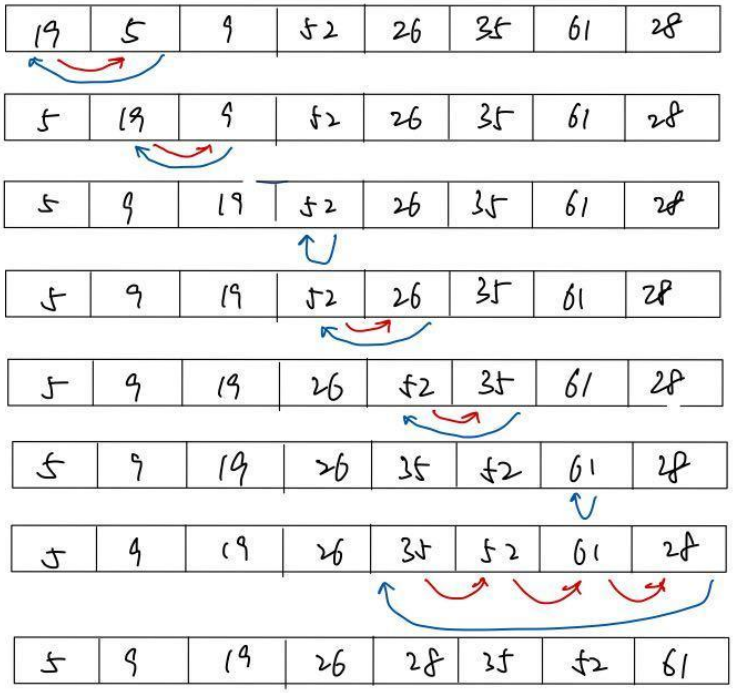
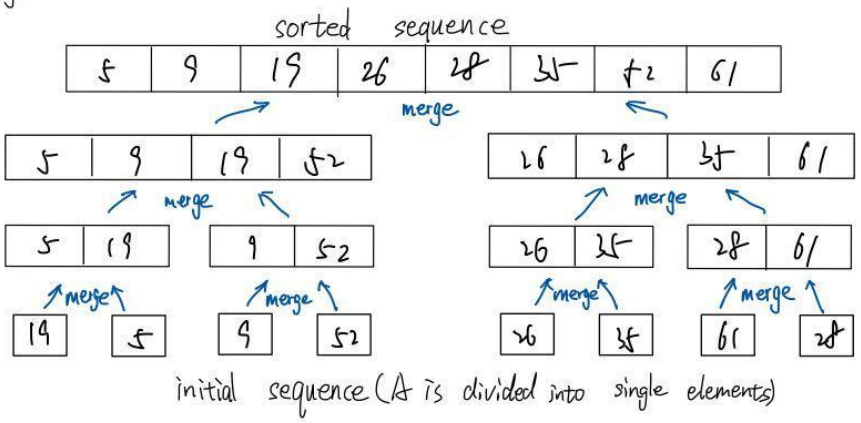
# Fall 2021 Homework 3 Solution

1.

a. insertion sort:



b. merge sort:




2.

a) SELECTION-SORT(A)

*n* = *A.length*

**for** *j* = 1 **to** *n* - *1*

*smallest* = *j*

**for** *i* = *j* + 1 **to** *n*

**if** *A[j]* < *A[smallest]*

*smallest* = *j*

exchange *A[j]* with *A[smallest]*

b) Loop invariant:

At the start of the outer **for** loop in line 1, the subarray $A[1...j-1]$ consists of the smallest $j$ - 1 elements in array $A[1...n]$, and this subarray is in sorted order.

c) After the first $n$ - 1 elements, the subarray $A[1...n-1]$ contains the smallest $n$ - 1 elements, sorted, and therefore element $A[n]$ must be the largest element.

d) The running time of the algorithm is $\Theta(n^2)$ for all cases.

3. For the maximum subarray problem, if we use divide-conquer, but instead of dividing the array into two halves, we equally divide it into three segments, how should the algorithm be modified? What is the running time of the new algorithm?

Solution: If we divide the original array A into 3 equal-sized sub-arrays S1, S2, and S3, we have 3 cases to consider in the combine phase of the divide-and-conquer algorithm:

(a) The max subarray starts from S1, end in S2: linear scan in S1 from tail to head and scan in S2 from head to tail and summation = $\Theta\left(\frac{2n}{3}\right) + \Theta(1)$

(b) The max subarray starts from S2, end in S3: linear scan in S2 from tail to head and scan in S3 from head to tail and summation = $\Theta\left(\frac{2n}{3}\right) + \Theta(1)$

(c) The max subarray starts from S1, end in S3: linear scan in S1 from tail to head and scan in S3 from head to tail and compute the sum of S2 and summation = $\Theta(n) + \Theta(1)$

So, we can get the recurrence formula: $T(n) = 3T\left(\frac{n}{3}\right) + \Theta\left(\frac{2n}{3}\right) + \Theta(1) + \Theta\left(\frac{2n}{3}\right) + \Theta(1) + \Theta(n) + \Theta(1) \Longrightarrow 3T\left(\frac{n}{3}\right) + \Theta(n)$. The result is $T(n) = \Theta(n\,logn)$

4.

*PRINT(origin, destination):*

print("Move the top disk from rod", *origin*, "to rod", *destination*)

*MOVE(n, start, end):*

*mid_rod = 6 - start - end* *# mid_rod is the remaining rod*

**if** *n* == 1:

  *PRINT(start, end)*

**else**:

  *MOVE(n - 1, start, mid_rod)*

  *MOVE(1, start, end)*

  *MOVE(n - 1, mid, end)*

Note: The Tower of Hanoi problem is mainly understood in three steps:

i. Send the “cone” of $n$ - 1 from the start position to mid_rod

ii. Send $n$ - ($n$ - 1) == 1 (the lowest disk) to end

iii. Send the “cone” of $n$ - 1 from mid to end

iv. The above three steps are recursive, each time the “cone” moves will avoid the real end, until there are 2 disks left at the end to start the final movement.

5. For an array with $n$ elements, design a *divide-and-conquer* algorithm for finding both the minimum and the maximum element of this array using no more than $3n/2$ comparisons. (Pseudocode is required)

Solution:

If we want to get a min and max of an array A, we can divide it into two part and compute the max of min, respectively. We can then combine them to get the max and min of A, pseudocode as below:

```
struct minmax{
    min
    max
    minmax(A, B)
            min = A
            max = B
};

MIN_MAX(A, low, high)
    if  (low == high)  //  only one item
             return minmax(A[low], A[low])
    if  (high == low + 1) //  2 items
            if  A[low] > A[high]
                    return minmax(A[high], A[low])
            else
                    return minmax(A[low], A[high])

            //  if A have more than 2 items
            mid = (low + high) / 2
            minmax_left = MIN_MAX(A, low, mid)
            minmax_right = MIN_MAX(A, mid + 1, high)
            //combine the min
            if (minmax_left.min < minmax_right.min)
                    temp1 = minmax_left.min
```

```
else
        temp1 = minmax_right.min
//combine the max
if (minmax_left.max < minmax_right.max)
        temp2 = minmax_right.max
else
        temp2 = minmax_left.max
return minmax(temp1, temp2)
```

Let's analyze the running time: Because there are 2 recursive call and other operations can be done in O(1) time (mainly the comparison between the minmax_left and minmax_right), we have recursive formula: $T(n) = 2T\left(\frac{n}{2}\right) + \Theta(1)$. Using master method, we can have $T(n) = \Theta(n)$, which is in linear time. WLOG, if n is a power of 2. We have $T(n) = 2T\left(\frac{n}{2}\right) + 2$ (2 comparisons). We can get that $T(n) = \frac{3n}{2} - 2 \leq \frac{3n}{2}$. Hence, this divide and conquer algorithm using no more than $\frac{3n}{2}$ comparisons.